Listing of the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Previously Presented) A method for implementing a global queue, wherein the queue has a head pointer, a tail pointer, and a plurality of elements each having a next pointer, the head pointer functioning as a next pointer of a last element of the plurality of elements when the queue is empty, the method comprising:

executing an add to end function for adding a new element to the tail end of the queue even when the queue is in a locked state immediately prior to execution of the add to end function in which the queue head pointer is null and a queue tail pointer does not point to the queue head pointer, the executing an add to end function including:

setting a next pointer of the new element to null;

as an atomic transaction, setting the queue tail pointer to point the new element while saving a location of the last element; and

setting the next pointer of the last element to point to an address of the new element by using the last element's saved location.

2. (Previously Presented) The method of claim 1 further comprising executing a locking function of the queue, the locking function including:

if the queue is not empty and not locked, as an atomic transaction, setting the head pointer to null and retaining a previous value of the head pointer; and

if the previous value of the head pointer is null and the queue is not empty, repeating the locking function.

3. (Previously Presented) The method of claim 2 wherein the queue is unlocked in a first situation in which the head pointer is not null and in a second situation in which the head pointer is null and the tail pointer points to the head pointer.

4. (Previously Presented) The method of claim 2 further comprising executing an add to front function for adding the new element to a front position of the queue, the add to front function including:

if the queue is empty, adding the new element to the tail end position of the queue; and

if the queue is not empty:

locking the queue;

setting the next pointer of the new element to the previous value of the head pointer; and

pointing the head pointer to the new element, thereby unlocking the queue.

5. (Previously Presented) The method of claim 2 further comprising executing a remove from front function to remove a front-most element from the queue, the remove from front function including:

locking the queue;

if the queue is not empty and an element occupying a front-most position of the queue has a next pointer that is not null, setting the head pointer to the address in the front-most element's next pointer; and

if the queue is not empty and the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer, thereby implicitly unlocking the queue.

6. (Previously Presented) The method of claim 5 further comprising, responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null, and pointing the head pointer to an element pointed to by the next pointer of the front-most element, thereby implicitly unlocking the queue.

7. (Previously Presented) The method of claim 2 further comprising executing a remove specific function to remove a target element from the queue, the remove specific function including:

locking the queue; and

if the queue is not empty:

traversing the queue to locate the target element;

if the target element's next pointer is not null and the target is not addressed by the previous value of the head pointer, setting the next pointer of an element previous to the target to point to an element pointed to by the target's next pointer; and

returning the head pointer to the previous value, thereby implicitly unlocking the queue.

8. (Original) The method of claim 7 further comprising:

if the target element's next pointer is not null and the target is addressed by the previous value of the head pointer, setting the head pointer to point to the element pointed to by the target's next pointer, thereby implicitly unlocking the queue; and

if the target's next pointer is null and the target is not addressed by the previous value of the head pointer, setting the next pointer of the element previous to the target to null.

9. (Original) The method of claim 8 further comprising:

if the target's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the target setting the tail pointer to point to the element previous to the target, or to point to the head pointer if the target is addressed by the previous value of the head pointer;

if the atomic compare and exchange was performed and failed:

waiting until the target's next pointer is not null;

if an element addressed by the target's next pointer is an only remaining element in the queue, setting the head pointer to point to the only remaining element, thereby implicitly unlocking the queue; and

if the element addressed by the target's next pointer is not the only remaining element in the queue, setting the next pointer of the element previous to the target to the address in the target's next pointer and setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue; and

if the atomic compare and exchange was performed and succeeded:

if the queue is not empty, setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue.

10. (Previously Presented) The method of claim 1 further comprising executing an empty function for removing each of the plurality of elements from the queue, the empty function including:

locking the queue; and

if the queue is not empty:

as an atomic transaction, pointing the tail pointer to the head pointer while retaining a previous value of the head pointer and the tail pointer, thereby implicitly unlocking the queue; and

by using the previous values of the head pointer and tail pointer, traversing a plurality of the elements which have been dequeued, and waiting for the next pointer of each element not addressed by the previous value of the tail pointer to become non-null.

11. (Previously Presented) A method for implementing a global queue in a multiprocessor environment, wherein the queue has a head pointer to point to a first element of the queue or to null if the queue is empty, a tail pointer to point to a last element of the queue or to the head pointer if the queue is empty, and a plurality of elements each containing a next pointer for pointing to a next element in the

queue or to null when the element occupies a last position in the queue, the head pointer functioning as a next pointer of the last element of the queue when the queue is empty the method comprising:

allowing a first processor to both add and remove elements from the queue;

allowing a second processor to only add new elements to the queue; and

executing an add to end function for adding the new element to the tail end of the queue, even when the queue is in a locked state immediately prior to execution of the add to end function in which the queue head pointer is null and a queue tail pointer does not point to the queue head pointer, wherein the executing an add to end function includes:

setting the next pointer of the new element to null;

as an atomic transaction, setting the tail pointer to point the new element, while saving a location of the last element; and

setting the next pointer of the last element to point to the address of the new element by using the last element's saved location.

12. (Previously Presented) The method of claim 11 further comprising executing an empty function for removing each element from the queue, the empty function including waiting until the head pointer is not null or until the queue is empty, and if the queue is not empty:

saving a value of the head pointer;

setting the head pointer to null;

as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the tail pointer; and

using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non-null.

13. (Previously Presented) The method of claim 11 further comprising executing a remove from front function for removing a front-most element from the queue, the remove from front function including waiting until the head pointer is not null or until the queue is empty, and if the queue is not empty:

if the front-most element's next pointer is not null, setting the head pointer to an address of the front-most element's next pointer;

if the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer.

14. (Previously Presented) The method of claim 13 further comprising, responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null, and pointing the head pointer to the element pointed to by the next pointer of the front-most element.

15. (Previously Presented) A system for implementing a global queue, wherein the queue has a head pointer to point to a first element of the queue or to null if the queue is empty and to function as a next pointer of a last element when the queue is empty, a tail pointer to point to the last element of the queue or to the head pointer if the queue is empty, and a plurality of elements each having a next pointer for pointing to a next element in the queue or to null when the element occupies a last position in the queue, the system comprising;

a first processor;

a plurality of instructions for execution on at least the first processor, the instructions including instructions for:

executing a locked state for the queue; and

subsequent to execution of the locked state, executing an add at end function for adding a new element to the tail end of the queue even when the queue is in a locked state in which the queue head pointer is null and a queue tail pointer does not point to the queue head pointer, the add at end function including setting the next pointer of the new element to null; as an atomic transaction, setting the tail pointer to point the new element, while saving a location of the last element; and setting the next pointer of the last element to point to the address of the new element by using the last element's saved location.

16. (Previously Presented) The system of claim 15 further comprising:

a second processor; wherein in the locked state only the first processor is allowed to remove elements from the queue.

17. (Previously Presented) The system of claim 16 further comprising instructions for executing an empty function for removing each element from the queue, the empty function comprising:

waiting until the head pointer is not null, or until the queue is empty; and

if the queue is not empty:

saving a value of the head pointer;

setting the head pointer to null;

as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the tail pointer;

and using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non null.

18. (Previously Presented) The system of claim 16 further comprising instructions for executing a remove from front function for removing a front-most element from the queue, the remove from front function comprising:

waiting until the head pointer is not null or until the queue is empty, and if the queue is not empty:

if the front-most element's next pointer is not null, setting the head pointer to the address in the front-most element's next pointer; and

if the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer.

19. (Previously Presented) The system of claim 18 further comprising instructions for, responsive to a failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null and pointing the head pointer to the element pointed to by the next pointer of the front-most element.

20. (Previously Presented) The system of claim 15 wherein the queue is unlocked in a first situation in which the head pointer is not null, and in a second situation in which the head pointer is null and the tail pointer points to the head pointer.

21. (Previously Presented) The system of claim 20 wherein the instructions for executing a locked state for the queue comprise:

instructions for locking the queue when the head pointer is null and the tail pointer does not point to the head pointer; and

instructions for executing a locking function for the queue, the instructions for executing a locking function comprising:

if the queue is not empty and not locked, as an atomic transaction, setting the head pointer to null and retaining a previous value of the head pointer; and

if the previous value of the head pointer is null and the queue is not empty, repeating the locking function.

22. (Previously Presented) The system of claim 15 further comprising instructions for executing an add to front function, wherein the new element is added to a front position of the queue, the instructions for executing an add to front function comprising instructions for:

if the queue is empty, adding the new element to a last position of the queue; and

if the queue is not empty:

locking the queue;

saving a previous value of the head pointer;

setting the next pointer of the new element to the previous value of the head pointer; and

pointing the head pointer to the new element, thereby unlocking the queue.

23. (Previously Presented) The system of claim 15 further comprising instructions for executing an empty function for removing each element from the queue, the instructions for executing an empty function comprising instructions for:

locking the queue; and

if the queue is not empty:

as an atomic transaction, pointing the tail pointer to the head pointer while saving a value of the head and tail pointers, thereby implicitly unlocking the queue; and

by using the saved values of the head pointer and tail pointer, traversing the dequeued elements and waiting for the next pointer of each element not addressed by the saved value of the tail pointer to become non null.

24. (Previously Presented) The system of claim 15 further comprising instructions for executing a remove from front function for removing a front-most element from the queue, the instructions for executing a remove from front function comprising instructions for:

locking the queue;

if the queue is not empty and the front-most element's next pointer is not null, setting the head pointer to an address in the front-most element's next pointer; and

if the queue is not empty and the front-most element's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the front-most element, pointing the tail pointer to the head pointer, thereby implicitly unlocking the queue.

25. (Previously Presented) The system of claim 24 further comprising instructions for, responsive to performance and failure of the atomic compare and exchange, waiting for the next pointer of the front-most element to become non-null and pointing the head pointer to the element pointed to by the next pointer of the front-most element, thereby implicitly unlocking the queue.

26. (Previously Presented) The system of claim 15 further comprising instructions for executing a remove specific function for removing a target element from the queue, the instructions for executing a remove specific function comprising instructions for:

locking the queue;

determining if the queue is not empty; and

if the queue is not empty:

traversing the queue to locate the target element; and

if the target element's next pointer is not null and the target element is not addressed by the previous value of the head pointer, setting the next pointer of an element previous to the target element to point to an element pointed to by the target element's next pointer, and return the head pointer to the previous value, thereby implicitly unlocking the queue.

27. (Previously Presented) The system of claim 26 further comprising instructions for:

if the target's next pointer is not null and the target is addressed by the previous value of the head pointer, setting the head pointer to point to the element pointed to by the target's next pointer, thereby implicitly unlocking the queue;

if the target's next pointer is null and the target is not addressed by the previous value of the head pointer, setting the next pointer of the element prior to the target to null; and

if the target's next pointer is null, as an atomic compare and exchange, if the tail pointer points to the target set the tail pointer to point to the element previous to the target, or to point to the head pointer if the target is addressed by the previous value of the head pointer.

28. (Previously Presented) The system of claim 27 further comprising instructions for, responsive to performance and failure of the atomic compare and exchange:

waiting until the target's next pointer is not null;

if an element addressed by the target's next pointer is an only remaining element in the queue, setting the head pointer to point to the only remaining element, thereby implicitly unlocking the queue;

if the element addressed by the target's next pointer is not the only remaining element in the queue, setting the next pointer of the element previous to the target to the address in the next pointer of the target and setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue; and

if the atomic compare and exchanged was performed and succeeded:

if the queue is not empty, setting the head pointer to the previous value of the head pointer, thereby implicitly unlocking the queue.